Ramanarayanan, S. C.; Manthey, K.; Juurlink, B.

High performance CCSDS image data compression using GPGPUs for space applications

Ramanarayanan, S. C.; Manthey, K.; Juurlink, B. (2015). High performance CCSDS image data compression using GPGPUs for space applications. PARS: Parallel-Algorithmen, -Rechnerstrukturen und -Systemsoftware, 32(1), pp. 49-58. https://hdl.handle.net/20.500.12116/1928

High performance CCSDS image data compression using GPGPUs for space applications

Sunil Chokkanathapuram Ramanarayanan[a], Kristian Manthey[b], Ben Juurlink[a]

[a]Technische Universität Berlin
Embedded Systems Architecture
Einsteinufer 17
10587 Berlin, Germany

[b]German Aerospace Center (DLR)
Institute of Optical Sensor Systems
Optical Sensors and Electronics
Rutherfordstraße 2
12489 Berlin, Germany

Abstract: The usage of graphics processing units (GPUs) as computing architectures for inherently data parallel signal processing applications in this computing era is very popular. In principle, GPUs in comparison with central processing units (CPUs) could achieve significant speed-up over the latter, especially considering data parallel applications which expect high throughput. The paper investigates the usage of GPUs for running space borne image data compression algorithms, in particular the CCSDS 122.0-B-1 standard as a case study. The paper proposes an architecture to parallelize the Bit-Plane Encoder (BPE) stage of the CCSDS 122.0-B-1 in lossless mode using a GPU to achieve high throughput performance to facilitate real-time compression of satellite image data streams. Experimental results are furnished by comparing the performance in terms of compression time of the GPU implementation versus a state of the art single threaded CPU and an field-programmable gate array (FPGA) implementation. The GPU implementation on a NVIDIA® GeForce® GTX 670 achieves a peak throughput performance of 162.382 Mbyte/s (932.288 Mbit/s) and an average speed-up of at least 15 compared to the software implementation running on a 3.47 GHz single core Intel® Xeon™ processor. The high throughput CUDA implementation using GPUs could potentially be suitable for air borne and space borne applications in the future, if the GPU technology evolves to become radiation-tolerant and space-qualified.

1 Introduction

The spatial as well as the spectral resolution of air borne and space borne image data increases steadily with new technologies and user requirements resulting in higher precision and new application scenarios. On the technical side, there is a tremendous increase in data rate that has to be handled by such remote sensing systems. While the memory capacity requirements can still be fulfilled, the transmission capability becomes increasingly

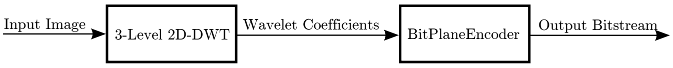


Figure 1: General Schematic of the CCSDS 122.0-B-1 Encoder [Con07].

problematic. The communication bandwidth is always an expensive entity and is limited by the radio channel and also the visibility of the ground control station (GCS) from the orbit. Moreover, the key requirement is to compress huge amounts of data in real-time and transmit them in as little time as possible to the GCS. These stringent requirements mandate a high performance real-time on-board compressor to attain high compression throughput of the orders of $^{\text{Mbyte}}/_{\text{s}}$. The generic commercial off-the-shelf (COTS) CPU technologies cannot be used for reliability reasons, space qualification criteria, and high throughput requirements. GPUs, by virtue of thread level parallelism (TLP) easily cater to the requirements of the massive data parallel image-processing applications. However, GPUs haven't yet been space-qualified as of now.

The Consultative Committee for Space Data Systems (CCSDS) compression standard exhibits data parallelism inherently in its encoding stages. It comprises of 2 major stages namely, discrete wavelet transform (DWT) and BPE as illustrated in the Fig. 1. The BPE stage is as computationally demanding as the DWT or more depending on the input images. This paper focuses on parallelizing the BPE stage of the CCSDS 122.0-B-1 standard using GPU. The CCSDS 122.0-B-1 compression standard operates in lossy/lossless modes based on the quality parameters. The lossless mode is expected to run through the entirety of the algorithm without exiting intermediately, to produce the worst case execution time in comparison to the lossy mode which could be configured to exit at different stages based on the quality parameters. This paper focuses on the lossless mode to analyze the worst case execution time of the encoder. Also, it could also be safe enough to assume lossless or near-lossless compression to be obligatory in several cutting-edge scientific missions which refuse the ideology of abiding by lossy compression.

The paper is organized as follows. Chapter 2 describes related work concerned to the image processing solutions for space applications, prior work pertaining to comparable compression standards on General-Purpose Computation on Graphics Processing Units (GPGPUs) to understand upfront about the state of the art technologies and research. Chapter 3 explains the fundamentals of CCSDS 122.0-B-1 image data compression standard. Chapter 4 proposes the design and development of a GPGPU-based BPE stage of CCSDS 122.0-B-1 compressor. It explains the various design decisions; porting, parallelization, thread mapping strategies and optimizations done for the GPGPU solution. Chapter 5 comprehensively analyzes the performance benchmarks of the GPGPU implementation against the FPGA and host counterparts. Chapter 6 summarizes the results and discusses about the pros and cons of the GPGPU solution. Finally, chapter 7 summarizes the findings of the paper.

2 Related work

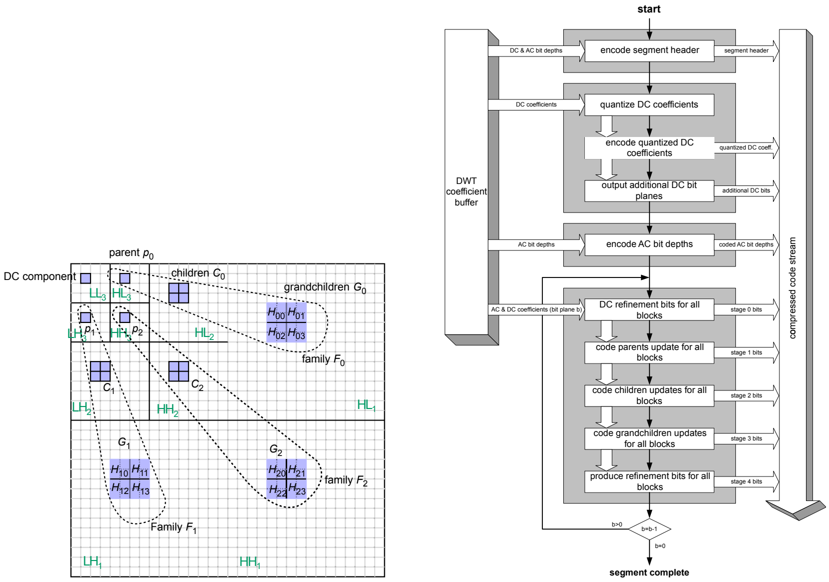
Many image compression standards such as JPEG2000 [Kur12], SPIHT [SLH11, LBM11] to name a few, have been tried on GPGPUs due to the presence of inherent data-parallelism. The DWT is an integral part of most of the image compression standards and is completely data parallel wherein every pixel could be independently processed. The following table 1 lists the prior DWT implementations on GPUs. Shifting focus to the CCSDS compression standards, [KAH^{+}12] proposes a GPGPU-based Fast Lossless hyper-spectral image compressor which achieves a throughput of 583.08 Mbit/s and a speed-up of 6 times in comparison with 3.47 GHz single core Intel® Xeon™ processor. Having seen significant research in the DWT implementations on GPGPU, the paper focuses on parallelizing the BPE stage of the CCSDS 122.0-B-1 standard in lossless mode. This paper also comparatively analyzes the GPGPU implementation with the hardware FPGA implementation of the CCSDS 122.0-B-1 standard by [MKJ14]. The FPGA implementation achieves an average throughput of 238.274 Mbyte/s.

Related works	Speedup (X times w.r.t host)	Reference CPU configuration	GPU card
GPU-Based DWT Acceleration for JPEG2000[Mat09]	148	Intel Core i7, 3.2GHz, 3 × 2GB RAM	NVIDIA Geforce GTX295
A novel parallel Tier-1 coder for JPEG2000 using GPUs[LBM11]	100	Intel Core i7, 2.8GHz, 12GB RAM	NVIDIA Geforce GTX480
A GPU-Accelerated Wavelet Decompression System with SPIHT[SLH11]	158	2 Intel quad-core Xeon E5520, 2.27 GHz, 16GB RAM	NVIDIA Tesla C1060

Table 1: Wavelet transform implementations using GPGPUs

3 CCSDS 122.0-B-1 image data compression

CCSDS 122.0-B-1 image data compression standard [Con05, YAK^{+}05] is a single-band compression technique and has been recently used for image data compression on-board spacecraft. It can compress 16 bit signed and unsigned integer images in lossless as well as in lossy mode. At first, the DWT module applies a 3-level 2D-DWT on the input image. The lossless "Integer DWT" implementation is considered in this paper to ensure that the original image is perfectly reconstructed. The DWT module forms a hierarchy of wavelet coefficients as shown in Fig. 2a. A *block* is a group of one DC coefficient and the 63 corresponding AC coefficients (3 parents, 12 children, 48 grandchildren). A block loosely represents a region in the input image. For the BPE, the blocks are further arranged into groups: A segment is a group of S consecutive blocks, where $16 \leq S \leq 2^{20}$. Segments are encoded independently and are further partitioned into *gaggles*, which is a group of $G = 16$ consecutive blocks. Once all the coefficients are grouped, the BPE starts to encode the image segment-wise (see Fig. 2b). Each segment starts with a *segment header* containing information about the current segment. After the segment header is written, the DC coefficients are quantized with a quantization factor q that depends on the wavelet



(a) A block consisting of a DC coefficient and 63 AC coefficients [Con05].

(b) Program and data flow of BPE [Con07]

Figure 2: Overview of the CCSDS 122.0-B-1 BPE

transform type and on the dynamic range of the wavelet coefficients. In a next step, differential pulse code modulation (DPCM) is applied on the quantized DC coefficients and is followed by Rice Coding. After all quantized DC coefficients are encoded, some additional DC bit-planes may be refined. The next step is to encode the bit-depth of the AC coefficients in each block with the same DPCM method. The BPE encodes the wavelet coefficients bit-plane-wise and in decreasing order. For each bit-plane, the encoding process is divided into stages 0–4. In stage 0, remaining bits of the DC coefficients are coded (DC refinement). Stage 1–3 encode the AC coefficients' sign and the position of the *significant bit*, which is the highest non-zero bit. Stage 1 refers to the refinement of the parents coefficients. The same procedure is applied to the children coefficients at stage 2 and to the grandchildren coefficients at stage 3. Stages 1–3 produce words which are first mapped to symbols which are then encoded with variable-length code (VLC). Once an AC coefficient is selected, Stage 4 encodes the AC coefficients' refinement.

4 Design

The goal of this paper is to parallelize the segment-wise BPE stage of the CCSDS 122.0-B-1 standard. Each segment could be executed mutually exclusive of the other. The BPE loop nest could be visualized wherein the DWT processed pixels are partitioned into NUM_SEGMENTS segments. The 1D loop is mapped onto a 1D thread grid. Each thread

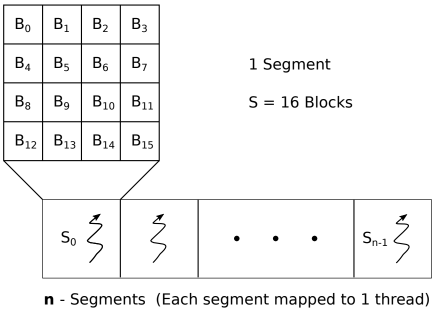


Figure 3: Thread mapping for the segment-wise BPE on GPU

processes one segment of the encoder as illustrated in Fig. 3. Each segment comprises of S blocks where $S = 16$ is chosen as a fixed parameter for this architecture. The reasons for this configuration shall be explained subsequently.

4.1 Thread mapping

Compute Unified Device Architecture (CUDA) allows to split the work in terms of threads, wherein a group of threads constitutes a thread block and further, a group of thread blocks forms a thread grid. The choice for these thread configurations is directly dependent on the loop index which is in this case the number of segments NUM_SEGMENTS. Also, the NVDIA architecture influences the potential size of the thread block. The threads within a thread block are further divided into groups of 32 threads called *warps* which are eventually scheduled by the *warp scheduler* onto the streaming multiprocessors (SMX). The NVIDIA GK104 SMX Kepler architecture has 4 warp schedulers to pick 4 active warps per clock cycle and dispatches them to the execution units. Hence it is always customary for each thread block to have at least 4 warps to ensure peak utilization of the SMX. Hence the number of threads within a thread block is chosen to be $32 \times 4 = 128$. Consequently, the number of thread blocks would be as follows.

```
int blocksPerGrid = (NUM_SEGMENTS/128) + 1;
```

4.2 Choice of encoder parameters

The CCSDS 122.0-B-1 standard is configured with quality parameters to operate in lossless mode in order to analyze worst case behavior and to facilitate perfect reconstruction of the input images. Fig. 4 shows the impact of the segment size S on the compression efficiency in lossless mode. Lesser the bits per pixel consumed by the compressor, better the compression efficiency. It can be noted that the value of S has minimal or no impact on the compression efficiency. Since the encoder is ported onto a GPGPU and the BPE

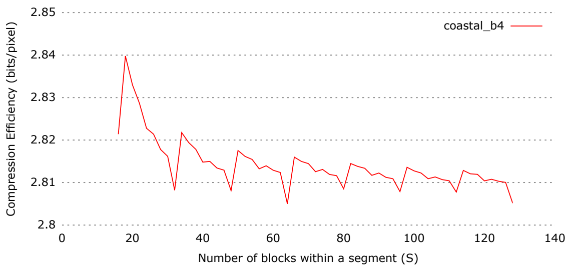


Figure 4: Impact of the segment size S on the compression efficiency

inherently has a lot of loop nests with loop index S, it is intuitive to reduce the value of S in order to reduce the branch latency considering the fact that GPUs perform poorly with branches due to branch divergence within the warps. Moreover, lesser the number of blocks within a segment, greater will be the total number of segments to process in the BPE stage. Therefore, minimizing the value of S results in achieving maximum possible value for NUM_SEGMENTS and thereby increasing the degree of parallelism. Hence, the segment size S is set to the least possible value of 16.

4.3 Concatenating the output bit-stream buffer

Since each segment is processed independently by different threads, the individual bit-streams generated by each thread have to be combined to produce the final output bit-stream buffer. Moreover, since dynamic allocation of the memory is impossible in GPU address space unlike the reference CPU implementation, the buffers have to be statically allocated up-front. In order to determine the size of the buffers, it is safe to assume that the compressed output bit-stream size shall not exceed the input barring the exceptions of high entropy images such as noise images. The input buffer size for each segment is $S \times 64 \times 4\,\mathrm{byte}$, as each segment contains S blocks and each block consists of 64 coefficients wherein the dynamic range of each wavelet coefficient does not exceed 20 bit. Hence the output bit-stream buffer size for each segment is also assumed to not exceed this limit for regular images. This concatenation process is performed on the host side after the CUDA kernel has completed its execution. Fig. 5 illustrates the process of the generation of the output bit-stream.

4.4 Optimizations

4.4.1 L1 Cache configuration

The NVIDIA GK104 SMX Kepler architecture offers a 64 kbyte unified memory subsystem useable as shared memory and also as L1 cache. In general, shared memory is

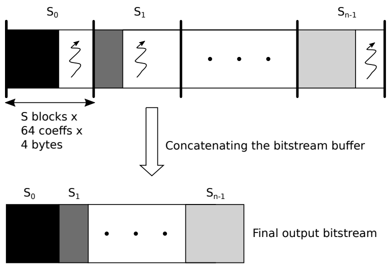


Figure 5: Output bitstream generation

configured for 48 kbyte and the remaining 16 kbyte behaves as L1 cache. By virtue of a lot of local variables in the BPE encoder, it is preferable to have a bigger L1 cache rather than the shared memory. Also, due to the fact that the input data set could not fit onto the shared memory, it was decided to use the bulk of the faster memory subsystem to act as L1 cache. The following CUDA API allows the programmer to set the preference for a 48 kbyte L1 cache.

```
// Prefer L1 cache of 48 kbyte instead of 16 kbyte
cudaFuncSetCacheConfig(encode, cudaFuncCachePreferL1);
```

4.4.2 Reduced Global memory accesses

The accesses to the global memory of the GPU is always expensive and keeping this in mind, repeated accesses to the same global memory variables were avoided by fetching them only once and efficiently rearranging the dependent computations.

5 Analysis

This section provides the results of the GPGPU implementation of the CCSDS 122.0-B-1 BPE and also compares it with the state of the art reference and FPGA implementations.

5.1 Methodology

5.1.1 Experimental setup

The reference implementation of the encoder shall be run on the state of the art 3.47 GHz single core Intel® Xeon™ for obtaining a host profile. The state of the art FPGA implementation from [MKJ14] is used for comparative analysis. The GPU used shall be NVIDIA® GeForce® GTX 670 launched in 2013 as a GPGPU platform to demonstrate the prototype. In a nutshell, the GPU card contains 7 NVIDIA Kepler GK104 SMX, each with 192 CUDA cores to sum up to a total of 1344 cores. Also, the presence of 2048 Mbyte of global GPU memory ensures a significant storage to fit the large input image data of the orders of $16\,\mathrm{k} \times 32\,\mathrm{k}$ pixel words.

5.1.2 Scope

The conformance single spectrum image set specified by the CCSDS 122.0-B-1 standard [Con07] is used to test the correctness of the encoder. Four images having different pixel bit depths namely $coastal\_b4$, ice_2kb4, foc and sar are chosen in this paper in order to have an extensive coverage of the encoder behavior. The encoded bit-streams are validated by decoding using a reference implementation of the CCSDS 122.0-B-1 decoder on the host machine. The input raw image data to the encoder and the output bit-stream from the decoder is checked for bit-wise match to ensure lossless reconstruction of the input image.

5.1.3 Profiling tools

The NVDIA command line profiler *nvprof* is used to measure the execution times of the CUDA GPGPU implementation running on NVIDIA GTX 670 GPU. The total execution time measured includes the CUDA kernel execution time as well as the host/GPU to/from memory transfers. The Linux clock/time API is used for obtaining the host profile of the CPU implementation. The FPGA profiling results are obtained using the cycle-accurate ModelSim™ simulator.

5.2 Impact of image sizes on performance

As already described in subsection 4.2, the parallelism is directly dependent on the value of the NUM_SEGMENTS. For normal sized images in the order 1024×1024 pixels, the NUM_SEGMENTS is not significantly high to utilize the GPU cores to good effect. Hence in order to improve the GPU utilization, the input image has to be big enough to ensure high occupancy of the cores. Therefore, for benchmark purposes, bigger images are generated by concatenating the entire original image as a linear buffer n times. Thus, the resultant image obtained is n times the original image size, thereby increasing the occupancy of the GPU cores by virtue of increased value of NUM_SEGMENTS. Four sets of images are created for $n = 1, 4, 16, 64$ for the performance analysis. Here, $n = 1$ represents the original image as is. Fig 6 shows that greater the image size is, better is the achieved throughput.

5.3 Performance Benchmarks

The throughput analysis is performed on the large sized images with $n = 64$ case on CPU, GPGPU and FPGA platforms. It could be observed in Fig. 7, that the GPGPU imple-

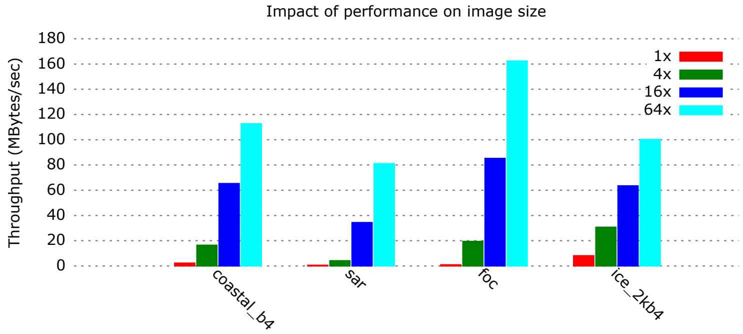


Figure 6: Comparison of Throughput based on Image sizes

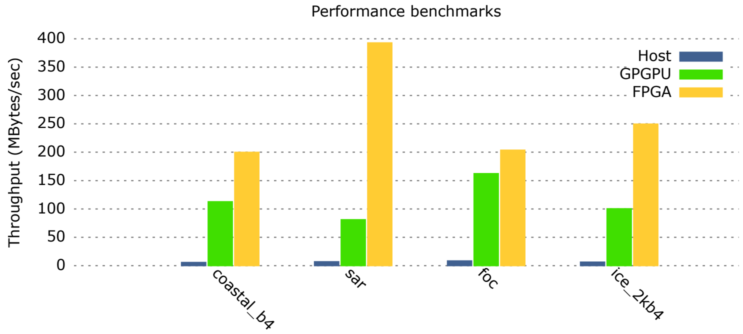


Figure 7: Throughput comparison between CPU, GPGPU and FPGA implementations

mentation achieves better throughput than the reference CPU implementation. However, the FPGA hardware implementation [MKJ14] achieves the best throughput in comparison with the CPU and GPGPU implementations.

6 Discussion

GPGPUs as the name suggests could be used for general purpose computation and isn't tightly bound to the application. GPUs could be used as a platform to run several different applications at different instants of time. The software programmability of the GPUs offers a high degree of flexibility in terms of application adaptation. GPUs also offer good backward compatibility. If an algorithm changes, the new software could run on older chip sets. The current GPU technologies have not proven to be space-qualified and radiation-tolerant. But, GPUs could easily be used in low Earth orbits and aeronautics.

7 Conclusion

The high-resolution satellite imaging systems require real-time image compressors onboard due to limited storage to store uncompressed raw data and/or also to conserve communication bandwidths. Hence it is mandatory for the image compressors to guarantee high compression throughput in the order of several $^{\mathrm{Mpx}}/_{\mathrm{s}}$. GPUs owing to their massive number of computing cores is investigated as a potential architecture platform to run the CCSDS 122.0-B-1 image data compression standard in lossless mode. The BPE stage of the standard was parallelized on GPU to satisfy the high compression throughput requirements. The parallelized BPE achieved an average speed-up of 16.718 times the host CPU implementation. The GPGPU solution is approximately 2.59 times slower in comparison to the state of the art hardware FPGA solution. This paper explores the possibilities of usage of GPU technologies for space applications provided they become radiation-tolerant and space-qualified in the future.

References

[Con05] Consultative Committee for Space Data Systems (CCSDS). Image Data Compression - Blue Book, 2005.

[Con07] Consultative Committee for Space Data Systems (CCSDS). Image Data Compression - Green Book, 2007.

[KAH+12] D. Keymeulen, N. Aranki, B. Hopson, A. Kiely, M. Klimesh, and K. Benkrid. GPU lossless hyperspectral data compression system for space applications. In *2012 IEEE Aerospace Conference*, pages 1–9. IEEE, March 2012.

[Kur12] Krzysztof Kurowski. Graphics processing unit implementation of JPEG2000 for hyperspectral image compression. *Journal of Applied Remote Sensing*, 6(1):061507, June 2012.

[LBM11] Roto Le, Iris R. Bahar, and Joseph L. Mundy. A novel parallel Tier-1 coder for JPEG2000 using GPUs. In *2011 IEEE 9th Symposium on Application Specific Processors (SASP)*, pages 129–136. IEEE, June 2011.

[Mat09] J Matela. GPU-Based DWT Acceleration for JPEG2000. In *MEMICS 2009 Proceedings*, pages 136–143, Brno, 2009.

[MKJ14] Kristian Manthey, David Krutz, and Ben Juurlink. A new real-time system for image compression on-board satellites. *ESA/CNES On-Board Payload Data Compression (OBPDC), Venice, Italy*, 2014.

[SLH11] Changhe Song, Yunsong Li, and Bormin Huang. A GPU-Accelerated Wavelet Decompression System With SPIHT and Reed-Solomon Decoding for Satellite Images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 4(3):683–690, September 2011.

[YAK+05] Pen-Shu Yeh, P. Armbruster, A. Kiely, B. Masschelein, G. Moury, C. Schaefer, and C. Thiebaut. The new CCSDS image compression recommendation. In *Aerospace Conference, 2005 IEEE*, pages 4138–4145, March 2005.